Search Space Reduction in Hyperparameter Optimization for Synthetic Tabular Data Generation

Lucas Ferreira Areias de Oliveira, Angelo Gaspar, Diego Kreutz, Kayua Oleques Paim Rodrigo Mansilha, Dionatan Ricardo Schmidt

[1]UNIPAMPA

Abstract. *Hyperparameter optimization in modern Artificial Neural Networks (ANNs) imposes significant challenges in terms of algorithmic and computational complexity, especially in tasks such as malware detection, behavioral analysis, and threat classification. This work proposes a three-stage methodology aimed at substantially reducing the search space during the HPO process. To validate the proposed methodology, we use the binary classification metrics Recall and F1-score applied to an ANN, leveraging three Android malware datasets. The results show a 91-fold reduction in the number of evaluated configurations compared to Grid Search in the context of synthetic tabular data generation for malware detection, demonstrating the potential of the proposed method for the efficient optimization of AI-based security models.*

1. Introduction

The appropriate choice of hyperparameters directly influences the complexity, behavior, and performance of machine learning models [Bischl et al. 2023], and is particularly critical in ANNs, which are highly sensitive to variations in these parameters. However, trial-and-error optimization can become infeasible due to the very large number of possible combinations.

To illustrate this complexity, we analyze MalSynGen, which uses multiple hyperparameters such as activation function, number of epochs, layer sizes, dropout rates, and optimization algorithms. Based on 12 analyzed papers, we estimate a total of approximately 36 trillion possible combinations. Executing all these configurations on a single dataset would require about 2 billion years, making exhaustive approaches impractical.

Several techniques have been proposed to mitigate this problem, such as *Grid Search*, *Random Search*, evolutionary algorithms, and Bayesian optimization [Bischl et al. 2023]. However, *Grid Search* and *Random Search* are inefficient in large search spaces [Bergstra and Bengio 2012], while Bayesian and evolutionary methods often demand high computational resources or a large number of iterations [Bischl et al. 2023]. In addition, the non-convex nature of hyperparameter optimization may result in suboptimal solutions [Luo 2016].

In this context, we propose a three-stage complementary approach that combines reference analysis, empirical experimentation, and adaptive refinement: (i) definition of initial ranges informed by multiple sources, including recurrent values in the literature, results from previous executions, and empirical knowledge about the problem domain; (ii) execution of exploratory tests over extreme and intermediate values in order to map promising regions of the search space; and (iii) iterative fine-tuning, focusing the exploration on the best-performing combinations observed so far.

The study validates the methodology using the MalSynGen tool [Nogueira et al. 2024], an ANN-based framework for synthetic tabular data generation in the context of Android malware detection, applied to two representative datasets from a total of eight, evaluating binary classification metrics and execution time.

2. Related Work

Table 1 presents the main related works on hyperparameter optimization in synthetic data generation models. It provides a categorization of scientific contributions from the state of the art related to hyperparameter optimization in ANNs and their variants for tabular data. This analysis is fundamental to understanding the search ranges used during hyperparameter optimization.

The first category includes studies that adopt manual approaches for configuration tuning, as shown in [Xu et al. 2024] and [Li et al. 2022], which focus on optimizing the generation of generic and medical tabular data, respectively. However, due to the exhaustive and low adaptability nature of the manual process, these approaches demand a considerably high computational time. As a consequence, they tend to use either more restricted search spaces or a reduced number of hyperparameters compared to the second category.

Table 1. Studies on hyperparameter optimization in ANNs

Article	Optimization Algorithm	Variation Range	Models
[Xu et al. 2024]	Manual tuning	Epochs: 100–5000; Batch: 20–300	CTGAN, TVAE
[Li et al. 2022]	Manual tuning	LR: 1e-4 (G) / 3e-4 (D); $\lambda = 1$; Batch: 32	TTS-CGAN
[Zhou et al. 2020]	Grid Search	Dilation: 8–64; Filters: 16–256; Kernel/Pool: 2–5; Dropout: 10–60%; Opt: various	LSTM-AE, TCN, GAN (MLP, LSTM)
[Basri et al. 2023]	Grid Search	Batch: 50–200; G/D LR: 1e-3–1e-5	CTGAN
[Nurhayati et al. 2021]	Grid/Random Search	Layers: 3; Units: 16–128; Act.: Sigmoid, ReLU; Opt: SGD, Adam...	GRU, LSTM, RNN
This work	Proposed method	Layer density: 32–4096; Epochs: 32–4096; Dropout: 0.05–0.6; Batch: 32–512	CGAN

The second category covers studies that employ automated optimization algorithms for hyperparameter search. Among them, [Basri et al. 2023] stands out by proposing a *framework* based on *Grid Search* for hyperparameter optimization in CTGAN variants. Furthermore, [Zhou et al. 2020] investigates the use of *Grid Search* and *Random Search* for optimizing different neural models, focusing on time series forecasting and cryptocurrency market analysis, respectively. It is worth noting that this group of works explores a search space that is larger or at least as large in terms of both the number of hyperparameters and the definition of value ranges, as they are not as heavily penalized by the manual tuning overhead.

In contrast to the analyzed works, this study proposes a method for hyperparameter optimization in ANNs that is applied to a more diverse set of hyperparameters and considers broader ranges for synthetic tabular data generation models in the Android malware domain.

3. Proposed Method

The three-stage approach (literature-based setup, edge-based exploration, and fine-tuning) aims to balance wide exploration of the hyperparameter space with computational efficiency, enabling an initially broad search followed by progressive refinement around the best-performing combinations.

3.1. Stage 1: Initial Definition of the Search Space

We carried out a systematic review following the guidelines of [Kitchenham et al. 2009] to identify critical hyperparameters and their initial search ranges. The search string combined the following key terms: ("hyperparameter tuning" OR "hyperparameter optimization") AND ("data augmentation" OR "synthetic data") AND ("deep learning" OR "machine learning") AND ("K-Fold cross-validation" OR "cross-validation") AND ("activation function") AND ("dropout rate" OR "dropout decay") AND ("layer size" OR "network depth" OR "epochs").

Articles published from 2018 onwards that presented at least one hyperparameter applicable to the context of this study were selected, totaling 12 analyzed papers. From this review, we defined initial intervals for a set of **13 hyperparameters**, including: generator neuron density (*dense_layer_sizes_g*), discriminator neuron density (*dense_layer_sizes_d*), number of epochs, number of folds (*k_fold*), training algorithm, activation function, discriminator dropout decay rate (*dropout_decay_d*), generator dropout decay rate (*dropout_decay_g*), batch size, latent dimension, initializer mean (*initializer_mean*), and initializer standard deviation (*initializer_deviation*). Based on these intervals, we defined an initial search space comprising 37,632 possible combinations (Table 3).

3.2. Stage 2: Initial Ranges and Edge-Based Sampling

From the lower and upper bounds found in the reviewed papers, we executed an exploratory approach based on *edge-based sampling* [Bergstra and Bengio 2012], in which combinations of hyperparameters are tested at the extremes of the search space.

Executions were carried out on two datasets selected as representatives of two distinct groups previously defined from a total of eight datasets, all belonging to the Android malware detection domain. Initially, configuration combinations were subjected to an exclusion filter to remove infeasible cases, according to the following criteria:

Evidence of network collapse: characterized by classification metrics close to zero or by progressive degradation of the discriminator performance across folds during training.

Interrupted or stalled executions: executions that presented critical failures such as deadlocks, out-of-memory errors, or fatal runtime errors were discarded. This choice is consistent with practices reported in other studies involving large-scale computational experiments [Salimans et al. 2016]. For these cases, we plan follow-up investigations to mitigate recurrent technical limitations.

After running the executions, we applied two criteria to disregard combinations that showed unstable or inconsistent results:

Cutoff 1 – F1-score: configurations with average *F1-score* below 0.7 across the five cross-validation folds were discarded. This metric balances precision and *Recall*, helping to avoid extreme trade-offs between false positives and false negatives.

Next, the ten best combinations were ranked, prioritizing those with the highest mean *Recall* values obtained on the synthetic data. Since *Recall* measures the model's ability to correctly identify positive samples (malware), it was prioritized due to its importance in scenarios where missed detections can lead to critical security failures.

3.3. Stage 3: Fine-Tuning and Iterative Search

Based on the best combinations identified in Stage 2, we performed a fine-tuning step, evaluating new combinations. In this phase, new lower and upper bounds for each hyperparameter were adjusted using the intervals from Stage 1 as reference, allowing a more focused search around the most promising configurations.

We applied a pruning step to remove ineffective configuration patterns observed in previous experiments. Combinations that presented recurrent patterns of poor performance were eliminated, such as cases in which the generator's dropout rate exceeded that of the discriminator. The remaining combinations were then evaluated in new executions.

Valid combinations again passed through the filtering criteria established in Stage 2. Only configurations with an average *F1-score* equal to or greater than 0.7 across the five cross-validation folds were kept. Finally, the ten best combinations were selected, prioritizing those with the highest mean *Recall* values, aiming at values as close as possible to 1.0.

4. Evaluation

4.1. Datasets and Artificial Neural Network

Table 2 presents the specifications of the eight datasets considered in this study. These datasets are frequently used in Android malware detection research [Revisão 2024]. In addition, they are part of a mapping effort conducted in a recent project spanning more than four years of research and development in collaboration with Motorola, a company specialized in Android-based devices.

Table 2. Datasets considered in this study

Group	Dataset	Samples	Features	Feature Type	Malware	Benign
1	AndroCrawl	96,744	142	Permissions	10,170	86,574
1	DefenseDroid PRS1	17,950	2,938	API Calls	5,975	11,975
1	DREBIN-215	24,507	215	Permissions, Intents, API Calls	9,476	15,031
1	KronoDroid Real Device	114,892	483	Permissions, Intents, API Calls	36,755	78,137
1	KronoDroid Real Emulator	63,991	483	Permissions, Intents, API Calls	28,745	35,246
2	DefenseDroid APICalls Closeness	10,476	21,997	API Calls	5,254	5,222
2	DefenseDroid APICalls Degree	10,476	21,997	API Calls	5,254	5,222
2	DefenseDroid APICalls Katz	10,476	21,997	API Calls	5,254	5,222

The datasets were grouped into two distinct sets based on a heatmap analysis of feature similarity between datasets. From these, two representative datasets were selected: AndroCrawl (group 1) and *defensedroid_apicalls_closeness* (group 2). Both underwent dimensionality reduction, limited to at most 200 features, using the Chi-Squared (χ^2) method, typically employed in feature engineering stages.

In addition to the datasets, we selected the MalSynGen tool [Nogueira et al. 2024], which implements ANNs for synthetic tabular data generation. The MalSynGen methodology follows a three-step workflow:

Selection step: performs data preprocessing including class balancing and K-fold cross-validation. In each iteration, the dataset is split into k training and validation subsets.

Training step: for ANN training, the tool implements the two evaluation strategies proposed by [Esteban et al. 2017], using two sets of classifiers. The first one, TRTS (*Training on Real data and Testing on Synthetic data*), trains classifiers on real data and evaluates them on synthetic data. The second one, TSTR (*Training on Synthetic data and Testing on Real data*), trains classifiers on synthetic data and tests them on real data.

Evaluation step: the evaluation considers both utility and fidelity of the generated data by computing classical classification metrics (*Accuracy*, *Precision*, *Recall*, *F1-score*, and AUC-ROC), as well as fidelity metrics such as cosine similarity, squared Euclidean distance, and mean squared error.

4.2. Results

The proposed method drastically reduced the number of tested combinations while preserving the quality of the generated models. Stage 1 carries out a controlled initial exploration of the search space using defined ranges for each hyperparameter in the malware domain (Table 3), resulting in 37,632 possible combinations. The three-stage approach reduced the total number of combinations by a factor of 91 compared to traditional *Grid Search*, while maintaining high *Recall* and *F1-score* values.

Figure 1 shows the distribution of synthetic *Recall* for the 100 best executions in each stage. In Stage 2 (blue), we observed a wide dispersion, with *Recall* values ranging from 0.1 to approximately 0.8, and a predominance of *F1-score* values below 0.5, reflecting instability and imbalance between precision and *Recall*. In contrast, Stage 3 (red) shows a distribution concentrated in the 0.8 to 0.9 range, indicating greater stability and generalization capability after hyperparameter refinement.

Overall, 81% of valid executions in Stage 3 achieved *Recall* above 0.7, compared to only 28% in Stage 2, highlighting a significant gain in consistency and performance enabled by the proposed approach.

Table 3. Value ranges considered for hyperparameters in Stages 1, 2, and 3

Hyperparameter	Stage 1 (Literature)	Stage 2	Stage 3
Generator neuron density (g)	32, 64, 128, 256, 512, 1024, 2048, 4096	32, 1024, 4096	512, 1024, 2048
Discriminator neuron density (d)	32, 64, 128, 256, 512, 1024, 2048, 4096	32, 1024, 4096	512, 1024, 2048
Number of epochs	32, 64, 128, 256, 512, 1024, 2048, 4096	32, 1024, 4096	500, 1000, 2000
K-fold	5	5	5
Training algorithm	Adam	Adam	Adam
Activation function	LeakyReLU	LeakyReLU	LeakyReLU
Dropout decay rate (d)	0.05, 0.1, 0.2 , 0.3, 0.4, 0.5, 0.6	0.05, 0.3, 0.6	0.05, 0.2, 0.4
Dropout decay rate (g)	0.05, 0.1, 0.2 , 0.3, 0.4, 0.5, 0.6	0.05, 0.3, 0.6	0.05, 0.2, 0.4
Batch size	32, 64, 128, 256	32, 256	32, 256, 512
Latent dimension	32	32	32
Number of layers	1–3	1	1
Initializer mean	-	0	0
Initializer standard deviation	-	0.125	0.125

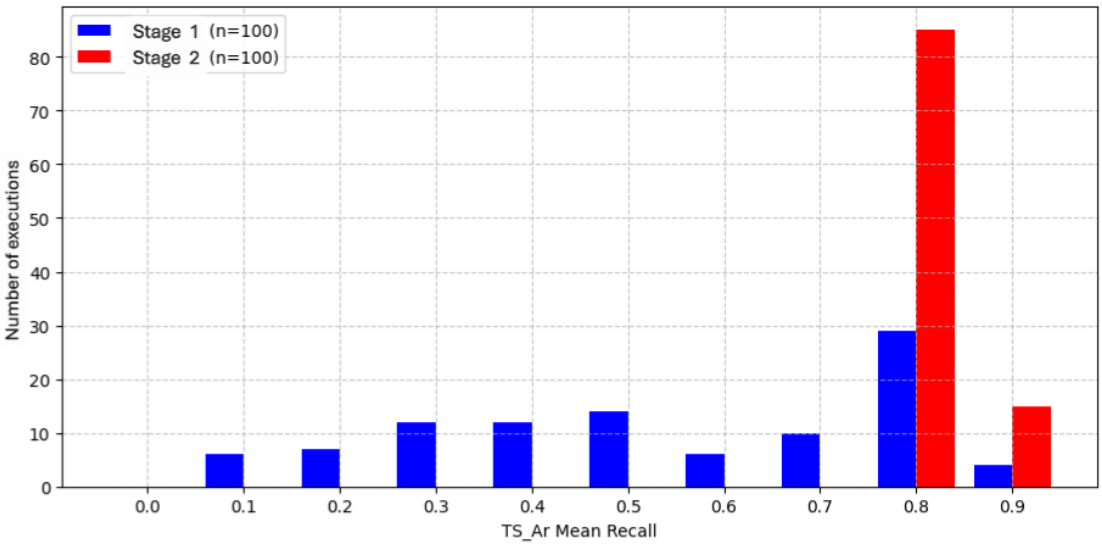

Figure 1. Distribution of *TS_Ar Mean Recall* for the 100 best executions in each stage.

5. Conclusion

This study presented a three-stage methodology for efficiently reducing the search space in ANN hyperparameter optimization, applied to synthetic tabular data generation for Android malware detection. The approach enabled a 91-fold reduction in the number of tested combinations compared to *Grid Search*, while preserving high-quality results. Final combinations were selected based on the best mean *Recall* values, prioritizing the model's ability to identify positive samples (malware), and more than 90% of valid executions remained after applying *F1-score* and *Max Mean Discrepancy* filters.

As future work, we plan to define a more robust comparative baseline and to extend the methodology to other data domains, broadening its applicability and efficiency. Moreover, Stage 3 will be explored in greater depth, focusing on the search for global minima through more refined analysis.

References

Basri, A., Hassan, A., and Yusof, N. (2023). A hyperparameter tuning framework for tabular synthetic data generation methods. *Journal of Computational Vision and Imaging Systems*, 5(1). Available online.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305.

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., et al. (2023). Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484.

Esteban, C., Hyland, S. L., and Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*.

Kitchenham, B., Brereton, P., Budgen, D., Turner, M., Bailey, M., and Linkman, S. (2009). Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15.

Li, Q., Zhang, Y., and Wang, J. (2022). Tts-cgan: Time-series conditional generative adversarial network for tabular data synthesis. *arXiv preprint arXiv:2206.13676*.

Luo, G. (2016). A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 5:1–16.

Nogueira, A., Paim, K., Bragança, H., Mansilha, R., and Kreutz, D. (2024). Malsyngen: redes neurais artificiais na geração de dados tabulares sintéticos para detecção de malware. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 129–136, Porto Alegre, RS, Brasil. SBC.

Nurhayati, R., Sari, A., and Pratama, M. (2021). Bitcoin prediction model using deep learning with hyperparameter optimization. In *2021 9th International Conference on Information Technology and Multimedia (ICIM)*, pages 394–399.

Revisão, A. (2024). Anonimizado para revisão. In *Anonimizado para Revisão*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*.

Xu, L., Skoularidou, M., Cuesta-Infante, A., and Veeramachaneni, K. (2024). Rigorous experimental analysis of tabular data generated using ctgan and tvae. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 15(4):1025–1033.

Zhou, K., Zhang, Y., and Wu, X. (2020). Time series forecasting and classification models using deep learning and attention mechanisms. *Sensors*, 20(24):7211.